METHOD, SYSTEM, AND PROGRAM FOR HANDLING DEVICE INTERRUPTS IN A MULTI-PROCESSOR ENVIRONMENT

BACKGROUND OF THE INVENTION

1. Field of the Invention

[0001] The present invention relates to a method, system, and program for handling device interrupts in a multi-processor environment.

2. Description of the Related Art

[0002] Computer systems may have a variety of devices including printers, network interfaces, keyboards, and disk drives which send or receive data within the system. These devices indicate the need to communicate with the rest of the system using what is commonly called an "interrupt." For example, a network controller might interrupt to signal the arrival of a new packet from the network or the successful transmission of a packet onto the network. Devices may signal the occurrence of other types of events using an interrupt.

[0003] The computer system typically has an interrupt handler which processes the event or events which triggered the interrupt. For example, many computer operating systems, such as Microsoft Windows®, Linux®, Unix®, etc., have an interrupt service routine (ISR) which responds to the receipt of an interrupt from various devices. (Microsoft and Windows are registered trademarks of Microsoft Corporation, Linux is a registered trademark of Linus Torvalds, UNIX is a registered trademark of The Open Group). These and other operating systems also execute various device driver programs that provide a software interface between the operating system and an associated device. A device driver includes device specific commands to communicate with and control the associated device. These device specific commands may be issued by an interrupt handler within the device driver (such as a device driver interrupt service routine (ISR)) for handling an interrupt generated by the associated device.

[0004] In many operating systems, upon receiving a device interrupt, the operating system identifies the device driver associated with the device that asserted the interrupt, and causes the device driver interrupt handler to be executed. Some operating systems, referred to as multiprocessing or parallel processing operating systems, permit running one or more programs on more than one processor such as a central processing unit (CPU). In computer systems having more than one CPU, various schemes have been proposed for the selection of the particular CPU to execute an interrupt handler. Some operating systems such as Microsoft Windows® have a task scheduler which selects the CPU to execute a task. In general the operating system task scheduler will select the CPU having the lightest load to execute the next task. Another technique routes all interrupts to a single processor for handling. Other systems utilize hardware-based "round-robin" algorithms which distribute interrupts evenly to processors of the system.

[0005] It is believed that there is a need in the art to provide improved techniques for executing interrupt handlers in a multiple processor computer system.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 illustrates a computing environment in which aspects of the invention are implemented.

FIG. 2 illustrates operations by a device driver to handle interrupts generated by an associated device in accordance with described implementations of the invention.

FIG. 3 illustrates operations performed by the operating system to handle an interrupt in accordance with described implementations of the invention.

FIG. 4 illustrates operations performed by the device driver to handle an interrupt in accordance with described implementations of the invention.

FIG. 5 illustrates operations performed by the device driver to handle an interrupt in accordance with described implementations of the invention.

FIG. 6 illustrates a computer architecture that may be used with the described embodiments.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0007] In the following description, reference is made to the accompanying drawings which form a part hereof and which illustrate several embodiments of the present invention. It is understood that other embodiments may be utilized and structural and operational changes may be made without departing from the scope of the present invention.

Device Driver Interrupt Handling

[0008] FIG. 1 illustrates an example of a computing environment in which aspects of the invention may be implemented. A computer 2 includes a plurality of central processing units (CPUs) 4a, 4b, 4c ... 4n, a volatile memory 6, a bus interface 8 on which devices communicate data and interrupts to the computer 2. A plurality of devices 10a, 10b...10n communicate data and interrupts to the computer 2 via a bus interface 8. The bus interface 8 may be implemented using any Input/Output (I/O) bus technology known in the art, such as the Peripheral Component Interconnect (PCI), Industry Standard Architecture (ISA), the Video Electronics Standards Association (VESA), Micro Channel Architecture (MCA), Extended ISA, and any other known bus technology known in the art. The devices 10a, 10b...10n may comprise any I/O device known in the art, such as storage devices (e.g., tape drive, hard disk drive, optical disk drive, memory card reader, etc.), storage controller, network controller, network adaptor card, video devices, printers, etc. The devices 10a, 10b...10n include one or more status registers 14a, 14b...14n that indicate, among other things, whether the device has asserted an interrupt on the bus 8. Although FIG. 1 only shows one bus 8, the computer 2 may include multiple busses to enable communication with the devices connected to such additional busses.

[0009] The computer 2 further includes an operating system 12, which may comprise any operating system known in the art, such as a Microsoft Windows® operating system, Linux®, a Unix® type operating system, etc. A bus driver 15 comprises a program that provides an interface between the operating system 12 and the bus 8 to enable communication between the operating system 12 and the devices 10a, 10b...10n that communicate on the bus 8. The

operating system 12 includes an interrupt service routine (ISR) component 16 that handles interrupt requests received from the devices 10a, 10b...10n transmitted across interrupt lines (not shown) of the bus 8. The operating system 12 further loads into memory 6 and executes at least one device driver 18a, 18b...18n for each device 10a, 10b...10n recognized by the operating system 12. The device drivers 18a, 18b...18n each include device specific code to enable communication between the operating system 12 and the devices 10a, 10b...10n. The device drivers 18a, 18b...18n each include an interrupt service routine (ISR) 20a, 20b...20n component to respond to an operating system ISR 16, and an interrupt handler (IH) component 21a, 21b, ... 21n to handle interrupt requests from the associated device 10a, 10b...10n. Each device driver 18a, 18b...18n further includes a maintenance component 24a, 24b...24n which can perform periodic tasks. The operating system ISR 16 utilizes a device driver list 22 that identifies all the loaded device drivers 18a, 18b...18n registered with the operating system 12.

[0010] The operating system 12 further includes a task scheduler 23 which decides which task or program will be run next and which one of the various CPU's 4a, 4b, 4c ... 4n will execute that program or task. In general the operating system task scheduler 23 will select the CPU 4a, 4b, 4c ... 4n having the lightest load to execute the next task.

[0011] The operating system 12 of the illustrated embodiment permits more than one task or program to run concurrently. In addition, each CPU 4a, 4b, 4c ... 4n can perform multitasking, that is, more than one program can run concurrently on each CPU 4a, 4b, 4c ... 4n. In practice, a CPU 4a, 4b, 4c ... 4n switches from executing one program to executing another program relatively quickly so as to give the appearance of simultaneously executing all of the programs assigned by the operating system 12 to that CPU 4a, 4b, 4c ... 4n. In the illustrated embodiment, the operating system 12 may permit preemptive multitasking in which the operating system 12 parcels out CPU time slices to each program. It is appreciated that other embodiments may use other types of multitasking such as cooperative multitasking in which each program can control an assigned CPU 4a, 4b, 4c ... 4n for as long as it needs it. If a

program is not using the assigned CPU 4a, 4b, 4c ... 4n, however, it can allow another program to use it temporarily.

[0012] The operating system 12 of the illustrated embodiment is also capable of multithreading, that is, executing different parts of a task or program, called threads, concurrently. The operating system 12 is responsible for allocating the system resources to competing processes, including competing threads and competing programs in a reasonable manner. This allocation is typically done by the task scheduler 23 in the illustrated embodiment.

[0013] In the illustrated embodiment, the selection of a particular processor on which to execute an interrupt handler task or process 21a, 21b ... 21n is preferably performed by the associated device driver 18a, 18b...18n rather than the operating system 12. Moreover, selection of a processor by the device driver 18a, 18b...18n is dynamic, based upon changing processor workloads. More specifically, each device driver 18a, 18b...18n temporarily pins its associated interrupt handler task 21a, 21b .. 21n to a particular processor or CPU 4a, 4b, 4c ... 4n based upon the level of usage of the CPU's in the system. All interrupt handlers 21a, 21b ... 21n associated with a device driver 18a, 18b...18n are executed by the particular CPU 4a, 4b, 4c ... 4n pinned by the device driver 18a, 18b...18n so long as usage of the CPU's 4a, 4b, 4c ... 4n does not change significantly. However, should the usage of the pinned CPU 4a, 4b, 4c ... 4n change significantly as compared to the usage of another CPU 4a, 4b, 4c ... 4n, a device driver 18a, 18b...18n can unpin the more heavily used CPU 4a, 4b, 4c ... 4n and pin a different, less heavily utilized CPU 4a, 4b, 4c ... 4n for subsequent execution of the interrupt handlers 21a, 21b ... 21n of the device driver 18a, 18b...18n.

[0014] For example, if the device driver for device 10a is device driver 18a, the device driver 18a can pin a particular CPU such as CPU 4b, for example, such that the interrupt handler 21a of device driver 18a is executed by CPU 4b each time device 10a generates an interrupt. However, should the usage of the pinned CPU 4b increase significantly as compared to the usage of another CPU 4a, 4c ... 4n, the device driver 18a for device 10a can unpin the

more heavily used CPU 4b and pin a different, less heavily utilized CPU such as CPU 4c so that CPU 4c executes the interrupt handlers 21a of the device driver 18a for each subsequent interrupt by device 10a until the device driver 18a pins a different CPU to execute the interrupt handlers 21a for the device 10a. Each of the other device drivers 18b, 18c ... 18n can dynamically pin the same or a different CPU of the CPU's 4a, 4b, 4c ... 4n to execute the interrupt handlers 21b, 21c ... 21n associated with each device driver 18b, 18c ... 18n.

[0015] FIG. 2 shows a diagram of an exemplary device driver 18a, 18b...18n spawning a maintenance thread 24a, 24b, 24c ... 24d which periodically determines the current utilization of each CPU 4a, 4b ... 4n in the system. The device driver 18a, 18b...18n can use this data to calculate the changing workload of each CPU 4a, 4b ... 4n over time. This utilization data may be stored for each device driver 18a, 18b...18n in a memory location 32a, 32b, 32c and 32d, for each CPU 4a, 4b, 4c ... 4n, respectively. The identity (ID) of the CPU 4a, 4b, 4c ... 4n currently selected or pinned to execute the interrupt handlers 21a, 21b ... 21n of a particular device driver 18a, 18b...18n may be stored in another storage location 34. In the illustrated embodiment, each device driver maintenance thread 24a, 24b ... 24n may have an associated data structure including memory locations 32a, 32b ... 32n to store the CPU utilization data obtained by that particular maintenance thread and a memory location 34 to store the CPU ID for the particular CPU 4a, 4b ... 4n pinned by that particular device driver maintenance thread.

[0016] As the workload of the CPU 4a, 4b ... 4n currently selected to execute the interrupt handler 21a, 21b ... 21n for the device driver 18a, 18b...18n increases, the device driver maintenance thread 24a, 24b .... 24n may opt to select a different, lesser utilized CPU on which its interrupt handlers 21a, 21b ... 21n should subsequently be executed. In which case, the device driver maintenance thread 24a, 24b, 24c ... 24d stores the ID of the new CPU 4a, 4b, 4c, 4d in the "Pinned CPU ID" storage location 34.

[0017] When the device 10a, 10b ... 10n of the device driver 18a, 18b...18n generates an interrupt, the operating system 12 enables an interrupt handler thread 21a, 21b ... 21n to be

subsequently spawned by the device driver 18a, 18b ... 18n of that device 10a, 10b ... 10n to handle that interrupt. The spawned interrupt handler thread 21a, 21b ... 21n is placed by the operating system 12 into the queue of the particular CPU 4a, 4b, 4c ... 4n which was pinned by the maintenance thread 24a, 24b ... 24n of the device driver 18a, 18b...18n for that device 10a, 10b ... 10n.

[0018] For example, FIG. 3 illustrates operations performed by an operating system interrupt service routine (ISR) 16 which may respond to the generation of a device interrupt. Upon receipt of an interrupt (block 100), the O/S ISR 16 determines (block 106) the identity of the particular device 10a ... 10n (FIG. 1) which generated the interrupt. The device driver interrupt service routine (ISR) 20a, 20b...20n of the particular device 10a, 10b ... 10n which generated the interrupt is then called (block 108) by the O/S ISR 16. The calling of the device driver ISR 20a, 20b ... 20n is represented by the spawning of the device driver ISR thread 20a, 20b ... 20n in FIG. 2.

[0019] FIG. 4 shows an example of operations performed by a device driver ISR thread 20a, 20b...20n which may respond to the O/S ISR 16. The device driver ISR 20a, 20b...20n reads (block 122) the identity of the CPU 4a, 4b, 4c ... 4n which has been pinned by the device driver maintenance thread 24a, 24b ... 24n. As previously mentioned in reference to FIG. 2, the maintenance thread 24a, 24b, 24c ... 24n stores the identity of the pinned CPU 4a, 4b, 4c ... 4n in a storage location 34 of a data structure for that particular maintenance thread. The device driver ISR thread 24a, 24b ... 24n reads the identity of the pinned CPU 4a, 4b, 4c ... 4n from this memory location 34 and calls (block 124, FIGs. 2, 4) the operating system task scheduler 23 (FIG. 2) to spawn an interrupt handler thread 21a, 21b ... 21n for the pinned CPU 4a, 4b, 4c ... 4n to execute the interrupt handler for that device driver 18a, 18b...18n. In the illustrated embodiment, the call to the operating system task scheduler 23 may be accomplished using an application program interface (API) call by the device driver ISR 20a, 20b ... 20n which pins the particular CPU 4a, 4b ... 4n to execute the interrupt handler 21a,

21b ... 21n as indicated in FIGs. 2 and 4. It is appreciated that tasks may be pinned to a particular CPU in a variety of approaches, depending upon the particular operating system.

[0020] In response to the pinned CPU interrupt handler API, the task scheduler 23 (FIG. 2) places the interrupt handler thread 21a, 21b ... 21n into the queue of the CPU 4a, 4b, 4c ... 4n pinned by the device driver 18a, 18b...18n to await execution. The next time the device 10a, 10b ... 10n associated with a device driver 18a, 18b...18n generates another interrupt, the operating system ISR 20a, 20b...20n 16 (FIG. 3) will again identify the device driver 18a, 18b...18n associated that device, call the device driver ISR 20a, 20b...20n (FIG. 4) of that device driver which will then cause another interrupt handler thread 21a, 21b ... 21n to be executed by the same pinned CPU 4a, 4b, 4c ... 4n. In this manner, all subsequent interrupt handlers 21a, 21b ... 21n for a particular device driver 18a, 18b...18n will be executed by the same pinned CPU 4a, 4b, 4c ... 4n until the device driver's maintenance thread 24a, 24b, 24c ... 24d pins a different CPU 4a, 4b, 4c ... 4n.

[0021] It is appreciated that there are various selection criteria which may be used for determining when to change the pinned CPU 4a, 4b, 4c ... 4n from one CPU 4a, 4b, 4c ... 4n to another. For example, a device driver 18a, 18b...18n might use a fixed margin or threshold to determine when to pin its interrupt handler 21a, 21b ... 21n to a different CPU 4a, 4b ... 4n. If the utilization of the current CPU exceeds the utilization of another available CPU by more than this margin, the driver 18a, 18b ... 18n might decide to switch its interrupt handler 21a, 21b ... 21n to the other, less utilized processor. For example, a margin of 15% may be selected for comparison purposes. If the utilization of the CPU 4a, 4b, 4c ... 4n currently pinned by the device driver 18a, 18b...18n is determined to be 50%, for example, and the utilization of another available CPU 4a, 4b, 4c ... 4n is determined to be less than 35%, this lesser utilized processor could then be pinned by the device driver 18a, 18b...18n in place of the CPU 4a, 4b, 4c ... 4n being utilized at 50%. It is appreciated that other margins or thresholds may be selected as well.

[0022] Monitoring of the utilization of the CPU's 4a, 4b, 4c ... 4n or other processors of the system is typically performed by the operating system 12. FIG. 2 depicts operations of an operating system CPU Monitor 130 which monitors the utilization of each of the processors or CPU's 4a, 4b, 4c ... 4n of the system. The measured utilization may be on an instantaneous basis, an average or weighted average basis, a trend basis or other statistical basis.

[0023] The maintenance thread 24a, 24b, 24c ... 24d spawned by the device driver 18a, 18b...18n queries the operating system CPU Monitor 130 (FIG. 2) to obtain the utilization data for each CPU 4a, 4b, 4c ... 4n available to execute the interrupt handlers 21a, 21b ... 21n of that device driver 18a, 18b...18n. The maintenance thread 24a, 24b, 24c ... 24d can store the utilization data obtained from the operating system 12 for each CPU 4a, 4b, 4c ... 4n directly into the associated memory locations 32a ... 32 for that maintenance thread. In addition, the maintenance thread 24a, 24b, 24c ... 24d can perform various statistical analyses on the utilization data including trend analyses, time averages etc. The results of the analyses may be stored in the associated memory locations as well. Thus, it is appreciated that a variety of statistical analyses may be utilized to determined whether the usage of the currently pinned CPU 4a, 4b, 4c ... 4n "significantly" exceeds that of another CPU 4a, 4b, 4c ... 4n.

[0024] If the maintenance thread 24a, 24b ...24n determines that there are multiple CPUs 4a, 4b ... 4n which are being utilized significantly less than the currently pinned CPU, then the device driver maintenance thread 24a, 24b ... 24n could select the CPU with the lowest utilization, for example. FIG. 5 shows an example of operations performed by a maintenance thread 24a, 24b, 24c ... 24d which may be periodically called by the device driver 18a, 18b...18n to monitor CPU usage and pin either the same or a different CPU 4a, 4b ... 4n depending upon the detected CPU usage. At the start (block 140) of the maintenance thread 24a, 24b, 24c ... 24d, a determination is made (block 142) as to the level of usage of the processors or CPU's 4a, 4b ... 4n of the system. This determination may be made in a variety of ways but typically would include calls to the operating system 12 such as the OS CPU Monitor 130 (FIG. 2) to obtain the processor utilization data. Next, the maintenance thread

24a, 24b ... 24n identifies (block 144) which of the multiple CPU's 4a, 4b ... 4n currently has the lowest utilization. In this example, a comparison is made (block 146) between the utilization of the currently pinned CPU 4a, 4b, 4c ... 4n to that of that lowest utilized CPU 4a, 4b, 4c ... 4n. If the currently pinned CPU 4a, 4b, 4c ... 4n is the lowest utilized CPU 4a, 4b, 4c ... 4n or if the utilization of the currently pinned CPU 4a, 4b, 4c ... 4n does not exceed that of the lowest utilized CPU 4a, 4b, 4c ... 4n by more than the predetermined margin, the pinned CPU 4a, 4b, 4c ... 4n does not change (block 148). As noted above, the value of the margin or threshold may vary. One example is a margin of 15%. Thus if the difference in utilizations of the currently pinned CPU 4a, 4b, 4c ... 4n and the lowest utilized CPU 4a, 4b, 4c ... 4n is less than 15%, the maintenance thread can leave the pinned CPU 4a, 4b, 4c ... 4n unchanged such that all subsequent interrupt handlers 21a, 21b ... 21n will continue to be pinned to the same CPU 4a, 4b, 4c ... 4n. Alternatively, if the difference in utilizations of the currently pinned CPU 4a, 4b, 4c ... 4n and the lowest utilized CPU 4a, 4b, 4c ... 4n is greater than 15%, the maintenance thread 24a, 24b .. 24n can unpin the current CPU 4a, 4b, 4c ... 4n (block 150) and pin (block 152) the new CPU 4a, 4b, 4c ... 4n determined to be the lowest utilized CPU 4a, 4b, 4c ... 4n. As a consequence all subsequent interrupt handlers 21a, 21b ... 21n for that particular device driver 18a, 18b ... 18n will be executed by the newly pinned CPU 4a, 4b, 4c ... 4n until the maintenance thread 24a, 24b ... 24n pins a different CPU 4a, 4b, 4c ... 4n because another CPU 4a, 4b, 4c ... 4n has a significantly lower utilization.

[0025] In the illustrated embodiment, each device driver 18a, 18b ... 18n has a maintenance thread 24a, 24b ... 24n to monitor CPU usage and a separate memory location 34 to pin a particular CPU to execute the interrupt handlers 21a, 21b ... 21n associated with the particular driver. It is appreciated that in other embodiments, a common maintenance thread may be utilized by two or more or even all device drivers 18a, 18b ... 18n such that a single, particular CPU 4a, 4b ... 4n is pinned to execute all the interrupt handlers 21a, 21b ... 21n of all the device drivers associated with the common maintenance thread until that common maintenance thread pins another, lesser utilized CPU. It is further appreciated that usage of a storage

location such as location 34 will vary between different embodiments. For example, in Microsoft Windows, this storage is transparent to the driver; the driver can merely tell the operating system once where to run its interrupt handling thread. After that, the driver need not specify the desired CPU until the driver wishes to switch to a different CPU.

Additional Embodiment Details

[0026] The described techniques for handling device interrupts may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" as used herein refers to code or logic implemented in hardware logic (e.g., an integrated circuit chip, Programmable Gate Array (PGA), Application Specific Integrated Circuit (ASIC), etc.) or a computer readable medium, such as magnetic storage medium (e.g., hard disk drives, floppy disks,, tape, etc.), optical storage (CD-ROMs, optical disks, etc.), volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, firmware, programmable logic, etc.). Code in the computer readable medium is accessed and executed by a processor. The code in which preferred embodiments are implemented may further be accessible through a transmission media or from a file server over a network. In such cases, the article of manufacture in which the code is implemented may comprise a transmission media, such as a network transmission line, wireless transmission media, signals propagating through space, radio waves, infrared signals, etc. Thus, the "article of manufacture" may comprise the medium in which the code is embodied. Additionally, the "article of manufacture" may comprise a combination of hardware and software components in which the code is embodied, processed, and executed. Of course, those skilled in the art will recognize that many modifications may be made to this configuration without departing from the scope of the present invention, and that the article of manufacture may comprise any information bearing medium known in the art.

[0027] In the described implementations, the interrupt handling implementations are included in a computer to handle interrupts from devices coupled to a bus enabling communication with

the computer. In alternative implementations, the interrupt handling implementations may be implemented in any type of electronic device communicating with other devices, such as a hand held computer, a palm top computer, a laptop computer, a network switch or router, a telephony device, a network appliance, a wireless device, etc.

[0028] In the described embodiments, certain operations were described as being performed by the operating system ISR 16 and device driver 18a, 18b...18n. In alterative embodiments, operations described as performed by the operating system ISR 16 may be performed by the device driver ISR 20a, 20b...20n, and vice versa.

[0029] In the described implementations, the devices communicated an interrupt signal for an I/O request over an interrupt line of the bus. In alternative implementations, the devices may signal an interrupt in a different manner than through a bus interrupt signal.

[0030] FIG 2 illustrates certain information maintained in computer memory. In alternative implementations, additional or different types of information may be maintained.

[0031] The illustrated operations of FIGs. 3-5 show certain events occurring in a certain order. In alternative embodiments, certain operations may be performed in a different order, modified or removed. Morever, steps may be added to the above described logic and still conform to the described embodiments. Further, operations described herein may occur sequentially or certain operations may be processed in parallel. Yet further, operations may be performed by a single processing unit or by distributed processing units.

[0032] In certain implementations, the device driver may be included in a computer system including a storage controller, such as a SCSI, IDE, etc. controller, that manages access to a non-volatile storage device, such as a magnetic disk drive, tape media, optical disk, etc. Such computer systems often include a desktop, workstation, server, mainframe, laptop, handheld computer, etc. In alternative implementations, the device driver embodiments may be included in a system that does not include a storage controller, such as certain hubs and switches.

[0033] In certain implementations, data may be transmitted on a cable connected to a port. Alternatively, embodiments may be configured to transmit data over a wireless network or connection, such as wireless LAN, Bluetooth, etc.

[0034] FIG. 6 illustrates one implementation of a computer architecture 200 of a computer such as the computer in FIG. 1. The architecture 200 may include a plurality of processors 202 (e.g., microprocessors), a memory 204 (e.g., a volatile memory device), and storage 206 (e.g., a non-volatile storage, such as magnetic disk drives, optical disk drives, a tape drive, etc.). The storage 206 may comprise an internal storage device or an attached or network accessible storage. Programs in the storage 206 are loaded into the memory 204 and executed by the processors 202 in a manner known in the art. The architecture further includes a network card 208 to enable communication with a network, such as an Ethernet, a Fibre Channel Arbitrated Loop, etc. Further, the architecture may, in certain embodiments, include a storage controller 209 to manage input/output (I/O) access to the data storage, where the storage controller 209 may be implemented on a storage card or device or integrated on integrated circuit components mounted on the motherboard. As discussed, certain of the network devices may have multiple network cards. An input device 210 is used to provide user input to the processors 202, and may include a keyboard, mouse, pen-stylus, microphone, touch sensitive display screen, or any other activation or input mechanism known in the art. An output device 212 is capable of rendering information transmitted from the processors 202, or other component, such as a display monitor, printer, storage, etc.

[0035] The foregoing description of various embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without

departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.